Prefetching Based on Web Usage Mining

Daby M. Sow, David P. Olshefski, Mandis Beigi, and Guruduth Banavar

IBM T. J. Watson Research Center
Hawthorne NY, 10532, USA
{sowdaby,olshef,mandis,banavar}@us.ibm.com

Abstract. This paper introduces a new technique for prefetching web content by learning the access patterns of individual users. The prediction scheme for prefetching is based on a learning algorithm, called Fuzzy-LZ, which mines the history of user access and identifies patterns of recurring accesses. This algorithm is evaluated analytically via a metric called *learnability* and validated experimentally by correlating learnability with prediction accuracy. A web prefetching system that incorporates Fuzzy-LZ is described and evaluated. Our experiments demonstrate that Fuzzy-LZ prefetching provides a gain of 41.5 % in cache hit rate over pure caching. This gain is highest for those users who are neither highly predictable nor highly random, which turns out to be the vast majority of users in our workload. The overhead of our prefetching technique for a typical user is 2.4 prefetched pages per user request.

1 Introduction

Large user perceived latency is a major problem in today's World Wide Web. Many factors contribute to this problem, including transmission latency, DNS name server lookups, TCP connection establishment, and start of session delays at the HTTP servers [3]. Conventional web caching techniques attempt to address part of this problem by temporarily storing recently accessed web content close to the user, on the client device or on a proxy server. These techniques work well when content is reused several times, potentially by several users. However, caching may not reduce latency when there is poor locality of reference and access to dynamic and personalized content.

A complementary approach to reducing latency is to effectively predict user access behavior and use this knowledge to prefetch content close to the user. Several research efforts [7,2,6] have studied aspects of prediction and prefetching. These efforts have focused on using the structure and relationship of content to predict user access behavior, and are described in detail in Section 2.

In this paper, we present a novel approach to prefetching that predicts future accesses by learning the usage patterns of individual users. The prediction algorithm mines access logs collected at or near the user. This allows us to focus on the access behavior of individual users rather than on the structure of content at any one Web server. The patterns generated by this technique capture a user's access behavior across multiple Web content servers. These patterns are

M. Endler and D. Schmidt (Eds.): Middleware 2003, LNCS 2672, pp. 262–281, 2003.

used to predict future accesses and to prefetch content to a caching proxy close to the user. The major advantages of this approach are: (1) since we identify patterns about individual users, we can provide differentiated services on a per-user basis, and (2) this is an intermediary-based approach that does not require modifications to the web server and is transparent to content providers.

Our prediction algorithm uses a new learning technique, called Fuzzy-LZ. This technique is an extension of the Lempel-Ziv compression algorithm [31]. Through analytical and experimental evaluation, we show that the predictive capability of this learning technique is proportional to the amount of randomness (which we term *learnability*) in the user's access behavior. We also evaluate the benefit of this learning technique when it is embedded within a web content prefetching test bed.

The following are the key contributions of this paper:

- *Fuzzy-LZ learning technique:* We propose a new learning technique for predicting individual user access patterns taking into account the fact that users access similar but not necessarily identical Web URLs[1]. We evaluate the effectiveness of this technique both analytically and experimentally, and show that the prediction accuracy of this technique is proportional to the randomness of a user's access behavior.
- *Learnability metric:* From complexity measures, we derive a metric called *learnability* which measures the randomness of the access behavior of an individual. This metric enables us to analytically estimate the accuracy with which a user's access behavior can be predicted. We studied a large population of users and found that the access behavior of most users was highly learnable (more than 80% of the users have a learnability measure between 0.65 and 0.85, on a scale of 0 to 1).
- *Prefetching system architecture and evaluation.* We evaluate the learning technique within an experimental web content prefetching system test bed. This test bed models a production enterprise proxy server by replaying the HTTP traffic over a specific time interval. For this workload, we obtained a gain of up to 41.5 % in cache hits over pure caching. The overhead of our prefetching technique for a typical user is only 2.4 pages per user request.

The rest of this paper is organized as follows. Section 2 provides a detailed account of related work in this area. Section 3 describes our Fuzzy-LZ technique for predicting user access behavior, by providing the background necessary to understand the LZ parsing algorithm. Section 3.2 then evaluates the Fuzzy-LZ algorithm, by first defininig the learnability metric and then by correlating the Fuzzy-LZ prediction accuracy with learnability. Section 4 describes our evaluation of a content prefetching system testbed that incorporates the Fuzzy-LZ technique and describes the experimental results. Section 5 points to some open issues and concludes.

[1] For example, all the news stories related to the Kenyan election on Dec 29th could have similar but not identical URLs on a news web site. This group of similar URLs is treated as a single unit. See Section 3 for details.

2 Related Work

Research related to this work falls into three categories: sequential prediction, web usage mining and prefetching. In this section, we survey the state of the art in each of these fields.

2.1 Sequential Prediction

Our work is built on the theoretical foundations of mature research fields of sequential prediction, information and rate distortion theories. Sequential prediction has developed techniques that predict the next event from a time ordered sequence of events. The work of Lempel and Ziv [12,31] on complexity and compression of finite sequences produced the LZ algorithm that can be considered a corner stone of this field. Feder et. al. [8] have shown that Lempel and Ziv's work can be used to design efficient and optimal finite state sequential predictors. Inspired by these results, Kumar et. al. [5] used the LZ algorithm to predict the location of a user in a home. Vitter and Krishnan [23] have investigated the use of the LZ algorithm for prefetching from a theoretical perspective. Lossy extensions of the LZ algorithm have been made in [16] for data compression purposes, with interesting links to rate distortion theory [4]. To the best of our knowledge, no one has applied such Lossy compression techniques to machine learning problems in general. Lossless compression algorithms have been successfully applied to a variety of machine learning problems. The links between learning and compression are not accidental. In fact, William of Ockham was the first one to state in the 14th century that the simplest explanations for arbitrary phenomena are always the most reliable. This principle, called the Occam razor principle has been widely applied in machine learning where simplicity is synonymous to conciseness [21].

2.2 Web Usage Mining

Srivastava et. al. [24] define Web Usage Mining as the application of data mining techniques to discover usage patterns from Web data, in order to understand and better serve the needs of Web-based applications. These applications include personalization, site modification, business intelligence, usage characterization, and site improvement applications. Several machine learning techniques have been successfully adapted for general applications. Most of these techniques have been applied to web server logs as opposed to proxy logs (see Section2.3 for the benefits and drawbacks). A notable exception is the work of Kerkhofs et. al. [15] that mines proxy log data. The authors attempt to find association rules [17] between the host parts of URLs to learn user browsing behaviors however they do not apply their technique to prefetching issues.

2.3 Prefetching

Prefetching has garnered a great deal of recent attention as a possible solution for reducing latency and bandwidth consumption on the web. In this section we survey the main theoretical and practical results in this area.

Jiang and Kleinrock [9] have studied the effect that prefetching has on network traffic from a queuing theoretical perspective. The main contribution of their work is the definition of a threshold on the likelihood that a document will be requested by a client. They show that prefetching documents with a likelihood below this threshold degrades the delivery performance of the network.

The Web Collector [27] and PeakJet2000 [18] (a commercial product) are systems that perform prefetching into a local cache based on the hypertext links of the previously downloaded pages. The drawback of this approach is the additional bandwidth load that their system imposes on the network. They improve their cache hit rate from 38.5% to 64.4% by prefetching 10 requests for every request made by the client [27]. Unfortunately, the traffic on the network triples. Our approach addresses this problem by reducing the number of prefetched pages based on information learned from access logs. We only prefetch an average of 2.4 extra pages for each request made by the typical web user and much less for highly predictable users and users with random patterns.

Duchamps [7] and Bestavros [1] separately propose keeping statistics on the "relatedness" of web pages and their embedded links, and use these statistics to make decisions on prefetching. This approach manages to ensure that 62.5% of prefetched pages are eventually used [7] but it completely ignores patterns across different hosts. For instance, this approach is unable to learn that requests to www.cnn.com and www.abc.com are correlated for a given user. Our approach complements Duchamps work and does not limit itself to predict only the accesses available via hyperlinks from the current page.

Davison [6] has proposed a technique that predicts a user's next web request by analyzing the content of the pages recently requested by the user. By prefetching the top 5 URLs predicted by this system, he shows a hit rate improvement over caching of up to 40% for an infinite cache. However, this approach adds a significant computational load on proxies supporting this scheme. Our approach is not as computationaly intensive and prefetches less objects per client request to get similar hit rate improvements.

Yang et. al. [29] have developed a prefetching technique that constructs association rules from web logs by building an n-gram model to predict future requests. They have reported improvements of 6% in fractional latency[2] with the addition of their prefetching module. Palpanas et. al. [20] also proposed a prefetching algorithm based on partial match prediction. Their simulations show that this technique is able to predict up to 23% of user's next web request but it has not been deployed on a real prefetching system.

Both Palpanas et. al. and Yang et. al. perform their mining on Web server logs, whereas we mine proxy server logs. The advantages of mining a proxy server log are: (1) Access patterns across multiple servers can be mined, and (2) to be deployed, the prefetching solution does not require any modifications to the origin servers. However, proxy server logs are more difficult to mine simply because there is much more variability in the requests made by a given user. Despite this, our approach yields a higher prediction accuracy on proxy logs.

[2] In [29] the authors define the fractional latency as the ratio of between the observed latency with a caching system and the observed latency without a caching system.

The work of Cohen et. al. [2] presents a different approach to prefetching. They propose a technique that aggregates pages at the web server into groups that they call volumes. In their approach, when a request is processed at the Web server, the Web server serves the request and pushes the entire volume containing the request. Clearly, in order to be widely deployed this approach requires a special protocol between Web servers and proxies for the distribution of volumes. Our approach operates exclusively at the client proxy and is easier to deploy.

Cohen and Kaplan [3] have shown that DNS query time, TCP connection establishment delays and start of session delays at HTTP servers are major causes of large latency. They propose methods for resolving DNS queries and opening up HTTP connections to content servers in anticipation of use. While their work does not predict the next URLs that a user will access, it complements our approach.

3 Predicting User Access Patterns

At the heart of all prefetching middleware lies a mechanism that infers the content that should be fetched in anticipation of its use. Such inferences can either be *informative* or *speculative*. In the former case, inferences are explicitly given to the middleware by an expert. In the latter case, inferences are derived by the middleware from historical data. In this section we focus on the problem of inferencing patterns from historical web usage data, for individual users, in order to learn their web browsing behavior and guide the prefetching middleware in the selection of content to prefetch. We call this problem *the web usage prediction problem*.

We cast this problem as a sequential prediction problem where we attempt to design predictors able to forecast the next URL that a user might access from the set of past URLs that this user has already accessed. Accordingly, inputs to these predictors are time ordered sequences of URLs that have previously been requested by the user. In our experiments, these inputs are obtained from an enterprise proxy server that stores access logs in the NCSA Common Log File Format [14]. We sought to develop an inferencing technique with the following properties:

1. The learning technique ought to be incremental and suitable for online learning, allowing the system to respond in real time to changes in user behavior.
2. The learning technique must be universal. By universal, we mean that it should be able to unconditionally track *any* user behavior. In practice, it should be able to track patterns of users without any knowledge of characteristics of individual users. With this property, we can deploy the same algorithm to a large set of users without having to statistically estimate the behavior of each individual user.
3. The learning technique must be well understood. This property allows us to extensively study the performance of the technique, investigate its optimality and understand its limitations.

4. The learning technique must be able to ignore small differences between similar URLs. This property is crucial for the mining of arbitrary URLs and differentiates this problem from many sequential prediction problems. Indeed, a large portion of URLs on the web are dynamic. The structure of many web sites like www.cnn.com evolves constantly. While a user might always access international news from this site on a daily basis, this user may not access the same URL more than once, simply because news stories are dynamic. Nevertheless, it is clear that the fact that this user periodically accesses the international section of this news site defines a pattern that should be identified by our predictors.

The desired output of the learning technique is a set of patterns of the form:

$$Condition \rightarrow Action$$

The condition is a trigger that defines the state in which a user must be for the pattern to be activated. In this work, the condition is an ordered sequence of URLs representing the last sites visited by the user. The action part of the pattern is a set of URLs. It represents the output of the prediction when the corresponding pattern is activated. In the rest of this section, we propose a novel technique to derive such patterns from user web logs for the web usage prediction problem.

3.1 The Fuzzy LZ Learning Technique

Sequential prediction problems have been addressed extensively in the literature and in particular in [8] from an information theoretical angle. One of the key results of this work is a proof of the efficiency of the well known Lempel-Ziv [12] parsing scheme for universal sequential prediction. It is shown in [8] that this parsing scheme can be used to design finite state predictors that are asymptotically optimal. Inspired by these strong results, we adapt and extend the LZ parsing scheme to solve the web usage prediction problem.

To formalize the problem, we follow [8]. We define U as an infinite sequence representing the sequence of URLs requested by a user. Elements of this sequence belong to Ω, the set of all URL visited. These elements are indexed with discrete time values: $U = \cdots, u_{-2}, u_{-1}, u_0, u_1, u_2, \cdots$. Observations of U are subsequences of U denoted U_i^j where i is the index of the first URL and j the index of the last. Fix the current time at n and imagine an observer that has sequentially received an arbitrary sequence of t URLs $U_{n-t+1}^n = u_{n-t+1}, \cdots, u_{n-1}, u_n$, and wishing to predict the next URL u_{n+1}. Our goal is to design a family of predictors $\{P^t\}, -\infty < t < +\infty$ performing such predictions with a minimal amount of prediction errors. $\{P^t\}$ is a family of functions $P^t : \Omega^t \rightarrow \Omega$, where Ω^t is the t-fold Cartesian product of Ω. The performance of each predictor is measured by its *accuracy*, which is its ability to correctly predict u_{n+1} from the past. Let $A_{\{P^t\}}^k$ be the accuracy of the entire family $\{P^t\}$ over the next k observations of URLs from the current time n. $A_{\{P^t\}}^k$ is equal to the fraction of correct predictions obtained during the testing phase of the learning process.

$$A^k_{\{P^t\}} = \frac{\sum_{i=0}^{k} 1(u_{n+i+1} = P^{t+i}(U^{n+i}_{n-t+1}))}{k}$$

where $1(\cdot)$ is the identity function[3]. Our goal can be translated into a quest for a family of predictors $\{P^t\}$ that maximizes $A^k_{\{P^t\}}$.

Original Incremental Lempel-Ziv Parsing. The Lempel-Ziv (LZ) parsing algorithm was introduced in 1976 in [12]. Soon after, the authors used it to design a universal compression technique [31] that became the standard algorithm for electronic file compression. The success of this algorithm is primarily due to its efficiency, its speed and its incremental nature. To describe how the algorithm works, consider the following example where we attempt to parse the following input sequence of URLs:

$$U^9_0 = a, b, c, a, c, b, b, a, c, b$$

where a, b, and c represent distinct URLs. The LZ algorithm parses this sequence into strings that have not appeared so far. For example, the input sequence U^9_0 is parsed as :

$$\tilde{U}^5_0 = a, b, c, ac, bb, acb,$$

Note that the main difference between U^9_0 and $\tilde{U}^5_0$ is the position of the comas. While elements of U^9_0 are URLs, elements of $\tilde{U}^5_0$ are now sequences of URLs. The key point to remember is that after every coma in $\tilde{U}^5_0$ the algorithm identifies the shortest string that has not been identified before in the parsing.

The LZ parsing has two important properties that allows us to use it for sequential learning problems:

1. As pointed out in [8], this parsing can be modeled as a process of growing a tree, where each new string is represented by a path in the tree. Such trees summarize the subset of all recurrent patterns that were identified by the LZ sequential parsing. Referring back to our desired output, any node in the tree represents the action part of a pattern and the list of ancestors of this node represents the condition part of the pattern. This ancestor list defines the state in which the user is believed to be before accessing the URL represented in the actual node. An example of tree obtained with this algorithm is shown in Figure 1.
2. This parsing can be used to design a family of optimal predictors. The derived trees do not track all the patterns present in the input sequences. Instead, they focus only on the ones with high probabilities. The patterns that are dropped have negligible probability of occurrence. For a formalization of the performance of this technique, please refer to [12,31,4,8].

The LZ parsing has the first three of the four criteria required for our learning solution as described in Section 3. The algorithm is incremental and suitable for online learning problems. It is well understood and has been studied extensively

[3] $1(X) = 1$ if X is true. Otherwise, $1(X) = 0$.

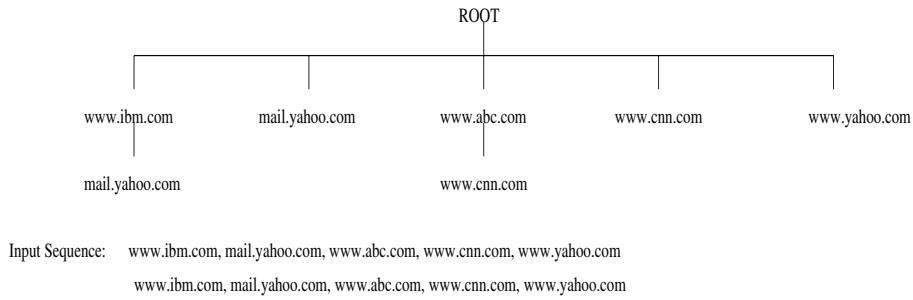

Fig. 1. Output of the original Lempel Ziv parsing algorithm

in the literature [4]. The technique is also universal. But it does not address the fourth criterion, namely the tolerance to small differences between URLs. In the next section, we extend this technique to address this fourth criterion.

Fuzzy-LZ: Fuzzy Extension of the LZ Parsing. While most sequential prediction schemes can be formulated from an information theoretical angle, the problem of mining web usage has a key characteristic that differentiates it from a typical sequential prediction problem. The URL sequence to predict typically evolves with time. As mentioned earlier, the structure of many web sites evolves constantly.

To address this problem, we have extended the LZ parsing scheme by allowing it to make approximate predictions. The main idea is to ignore the small variations in the structure of the URLs composing the input sequence. In essence, we group similar URLs into *clusters*. Instead of predicting individual URLs, we predict clusters of URLs. As a result, our predictors P_t output sets of URLs instead of individual URLs and the prediction accuracy $A^k_{\{P^t\}}$ becomes:

$$A^k_{\{P^t\}} = \frac{\sum_{i=0}^{k} 1(u_{n+i+1} \in P^{t+i}(U^{n+i}_{n-t+1}))}{k}$$

To find such predictors, we first define a semantic distance metric on the space of all possible URLs. Like any other distance metric, this metric quantifies the similarities between elements of its space. This metric is essentially computed by matching the different parts of URLs. Similarities between server names s_1 and s_2 are measured from the index of the last mismatch in their string representation, starting from the end of the strings. We start from the end of the strings because server names become more and more specific as we move from the end of the string to its beginning. Let $\hat{i}_{s_1,s_2}$ denote the index of the last mismatch between s_1 and s_2. Then the distance between s_1 and s_2 that we denote $d_{server}(s_1, s_2)$ is given by:

$$d_{server}(s_1, s_2) = \frac{max\{l(s_1), l(s_2)\} - \hat{i}_{s_1,s_2}}{max\{l(s_1), l(s_2)\}},$$

$l(\cdot)$ being the length function that returns the length of its string argument. To illustrate this, let $s_1 =$ www.yahoo.com and $s_2 =$ mail.yahoo.com. The index of the first mismatch starting from the end of the strings is 11 where the letters l and w do not match. Since $l(s_2) > l(s_1)$, $d_{server}(s_1, s_2) = \frac{l(s_2)-11}{l(s_2)} = \frac{3}{14}$.

Similarities between the path part of URLs are computed in a similar fashion. Let p_1 and p_2 be two valid path names. Let $\hat{j}_{p_1,p_2}$ denote the index of their first mismatch this time. Then the distance between p_1 and p_2 that we denote $d_{path}(p_1, p_2)$ is given by:

$$d_{path}(p_1, p_2) = \frac{max\{l(p_1), l(p_2)\} - \hat{j}_{p_1,p_2}}{max\{l(p_1), l(p_2)\}},$$

To illustrate this, let:
$p_1 =$ /doc/papers/03infocom.tex and $p_2 =$ /doc/papers/2003middleware.tex. The index of the first mismatch is 12, where the characters 0 and 2 do not match. Since $l(p_2) > l(p_1)$, $d_{path}(p_1, p_2) = \frac{l(p_2)-12}{l(p_2)} = \frac{29-12}{29} = \frac{17}{29}$.

Definition 1. *Let u_i, u_j be a pair of URLs. Let $s_{u_i}, p_{u_i}, q_{u_i}$ denote[4] respectively the server name part, the path part and the query part of the URL u_i. We define the semantic distance between u_a and u_b as a real number $d_{URL}(u_a, u_b)$ equal to:*

$$d_{URL}(u_a, u_b) = w_s d_{server}(s_{u_a}, s_{u_b}) + w_p d_{path}(p_{u_a}, p_{u_b}) + w_q d_{path}(q_{u_a}, q_{u_b})$$

where w_s and w_p and w_q are real coefficients.

$d_{URL}(\cdot, \cdot)$ is a distance metric[5]. It always returns a positive real number. This number measures how similar the arguments of $d_{URL}(\cdot, \cdot)$ are. The different weights w_s, w_p and w_q can be used to weigh the importance of the similarities between different parts of the URL. For the web usage prediction problem, we set w_s to 10, w_p to 1 and w_q to zero.

Using $d_{URL}(\cdot, \cdot)$, we have extended the LZ parsing scheme by adding fuzzyness in the derivation of the trees. To do this, we define a URL sphere of radius D, $\mathcal{S}(u_i, D)$, as the set of all URLs that are within D of u_i according to $d_{URL}(\cdot, \cdot)$. Instead of following the LZ parsing scheme that finds the shortest substring that has not been identified yet, *the key aspect of our proposed parsing is to find the shortest substring of spheres of radius D, that has not been identified yet, D being a parameter of the algorithm.* These spheres are dynamically generated as new URLs from the input sequence of URLs are read.

[4] There are more fields present in general URLs. To keep the discussion short, we focus only on the server name, the path and the query string but this definition can easily be extended to all parts of any well formed URL.

[5] To prove this statement, we would have to show that for any URLs u_a, u_b, and u_c, $d_{URL}(u_a, u_b) \geq 0$, $d_{URL}(u_a, u_b) = 0$ iff $u_a = u_b$, $d_{URL}(u_a, u_b) = d_{URL}(u_b, u_a)$ and finally that $d_{URL}(u_a, u_c) \leq d_{URL}(u_a, u_b) + d_{URL}(u_b, u_c)$. The proofs of each of these assertions are trivial, except for the last one, the triangular inequality. An easy way to prove it is to leverage the fact that the length of a match between two URLs is always larger than the length of the match between any of their prefix.

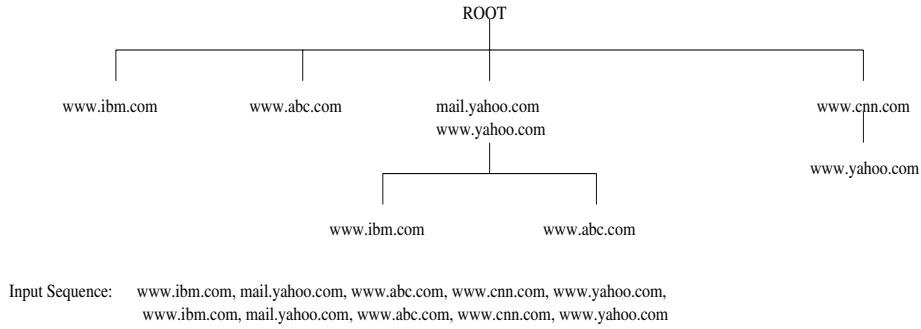

Fig. 2. Output of the Fuzzy-LZ Lempel Ziv parsing algorithm

An example of a tree obtained from this procedure is shown in Figure 2. D was set to 4/14 for this example. In contrast with Figure 1, the nodes of the tree shown in Figure 2 are clusters of URLs. In general, as the length of the input sequence increases, the number of nodes generated with the Fuzzy-LZ parsing technique is well below the number of nodes generated by the LZ technique. Hence, Fuzzy-LZ generates less patterns with higher accuracy but the price to pay for this accuracy improvement is the distortion introduced by the clusters.

3.2 Evaluation of Fuzzy-LZ

We have implemented Fuzzy-LZ and performed an extensive number of experiments to evaluate its effectiveness. A key index of the performance of a learning scheme is its *prediction accuracy* $A^k_{\{P^t\}}$. It is equal to the total number of correct URL predictions divided by the total number of requests made by the user. While the prediction accuracy might give an absolute view of the ability of the algorithm to make inferences, it does not take into account the characteristics of input sequences from which patterns are extracted. Indeed, regardless of the efficiency of our predictors, the accuracy of the end results of the predictions made is always bounded by the amount of patterns contained in the input sequence. In what follows, we quantify this amount with the concept of *learnability*. We define learnability and investigate its correlation with the prediction accuracy.

Analytical Evaluation: Learnability. The learnability of a sequence U_1^n is a real number assessing how difficult it is to predict U_1^n. This number is obtained by comparing the number of patterns identified in U_1^n to the number of patterns that would be present in a random sequence with length equal to the length of U_1^n. For example the periodic sequence $a, b, c, a, b, c, a, b, c, \cdots$ has a high value of learnability while most sequences obtained by a random generator would have very low values of learnability. This notion is similar to the notion of complexity of a sequence [13,8,12].

Definition 2. *Let $c^{U_1^n}$ denote the number of strings generated by the Fuzzy-LZ parsing of sequence U_1^n of n URLs. Then the learnability of U_1^n that we denote $L(U_1^n)$ is defined as:*

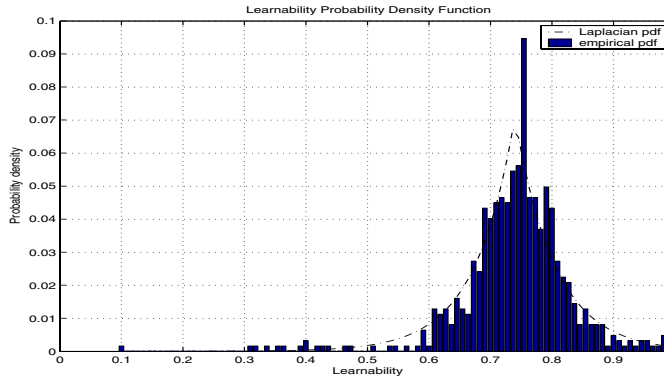

Fig. 3. Learnability probability density function. The mean and variance are respectively 0.7397 and 0.0084. The Laplacian density is obtained from these values

$$L(U_1^n) = 1 - \frac{c^{U_1^n} \log_2 c^{U_1^n} + c^{U_1^n}}{\log_2 \eta_{U_1^n}}, \tag{1}$$

$\eta_{U_1^n}$ denoting the number of different URLs appearing at least once in U_1^n.

Intuitively, $L(U_1^n)$ measures the amount of randomness in U_1^n. To see this, note that $c^{U_1^n} \log_2 c^{U_1^n} + c^{U_1^n}$ is equal[6] to the number of bits required to represent the sphere that contains U_1^n while $\log_2 \eta_{U_1^n}$ is the number of bits required to represent a random sequence of n URLs picked from the set of all URLs that appeared at least once in U_1^n. The ratio $\frac{c^{U_1^n} \log_2 c^{U_1^n} + c^{U_1^n}}{\log_2 \eta_{U_1^n}}$ is a compression ratio. Since this ratio is low for compressible sequences and high for random sequences, $L(\cdot)$ is high for compressible sequences and low for random sequences. Following the Occam Razor principle, we assume that sequences with low learnability are difficult to predict (see Section 2.1).

We have measured the learnablity for a large amount of log data containing the actions of 623 users. The resulting probability density function is shown in Figure 3. The shape of the density is Laplacian with mean 0.7397 and variance 0.0084. Figure 3 shows the distribution of learnability measured on the training set for all these users.

Experimental Results: Prediction Accuracy. From the probability density in Figure 3, we have identified 10 users spanning a range of learnability from 0.45 to 0.85. Our training set is composed of the actions of these users in a time period that starts on August 12 2002 at 12:00 AM and ends on August 15 2002 at 01:59 PM. Our testing set contains the actions that the same users made in a time period that starts on August 15 at 2:00 PM and ends August 16 at 11:59 PM. Using the testing set, we have evaluated the accuracy of the predictions for

[6] Please see [4] for a formal derivation of this result in a lossless setting.

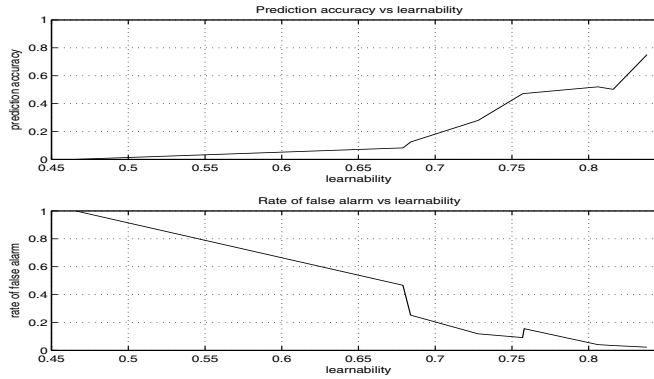

Fig. 4. Learning Effectiveness as a function of learnability

each of these 10 users by counting the number of correct predictions divided by the total number of requests made.

$$\text{prediction accuracy} = p_a = \frac{\text{total number of correct predictions}}{\text{total number of requests}}$$

We also measured the number of times that the predictor made an erroneous prediction. The rate at which such errors are made is called the rate of false alarms:

$$\text{rate of false alarms} = p_{f_a} = 1 - \frac{\text{number of correct prefetch}}{\text{total number of prefetch}}$$

Figure 4 shows the relationship between the prediction accuracy and the learnability. It also shows the relationship between the percentage of false alarms and the learnability. We notice that as the learnability increases, users become more predictable and the prediction accuracy increases while the percentage of false alarms decreases.

4 System Deployment on Experimental Test Bed

To demonstrate the effectiveness of Fuzzy-LZ, we have implemented the approach and deployed it on a system modeled after a real production enterprise proxy. The results presented here focus on the improvement in response time that Fuzzy-LZ is capable of providing for individual users. We compare these results with the results we obtained by using existing, start-of-the-art prefetching algorithms whose cache management is based on optimizing *aggregate* measures obtained for the set of all clients. We begin by describing our approach to designing an experimental test bed that models a production Internet proxy and then present the results obtained from our experiments.

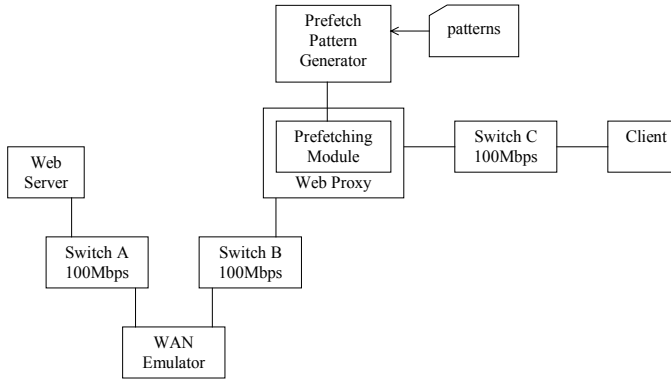

Fig. 5. Experimental test bed

4.1 Experimental Design

The testing environment consisted of four heterogeneous machines: a web server, a WAN emulator, a web proxy and one client machine (Figure 5). The proxy machine was an IBM Intellistation M Pro consisting of a 1.7 GHz CPU with 512 MB RAM. The server machine was an IBM NetVista consisting of a 1.8 GHz CPU with 512 RAM. The WAN emulator was an IBM Intellistation M Pro consisting of a 400 MHz CPU with 128 MB RAM and an IBM thinkpad 770ED consisting of a 266 MHz CPU with 96 MB RAM was used as the client machine. Both client and server machines ran RedHat 7.3, while the proxy ran Windows 2000 and the WAN emulator ran FreeBSD 4.6. Each computer had an 100/10 Mbps ethernet adapter and the three switches in Figure 5 were 100 Mbps fast ethernet switches from NetGear; switch A and B were model FS105 and switch C was a model FS108.

The WAN emulator software used was DummyNet [22], a flexible and commonly used FreeBSD tool. The WAN emulator simulated network environments with different network latencies. For our tests, we set the average round trip time to 200 ms. The latest stable version of the Apache HTTP server, V1.3.20, was executing on the server machine and was configured to run 255 daemons. The proxy machine was also running a version of Apache for Windows 2000, V1.3.26.

Although our test facilities hardware does not necessarily match that of our Internet proxy, we were able to develop traffic patterns in our testing environment that matched those observed by the production proxy server. Specifically, we created a file system on our backend web server that was a shadow of the file set requested at the production proxy server. We also created a traffic generator that created client loads based on the loads observed in the web log files at the production proxy server. The process for achieving this is outlined as a series of tasks in Figure 6.

Step 1: We downloaded the set of web logs from the production proxy server that were collected during the week of August 12 to 16, 2002. This set of five files contained a total of 4,109,814 web log entries in NCSA CLF format.

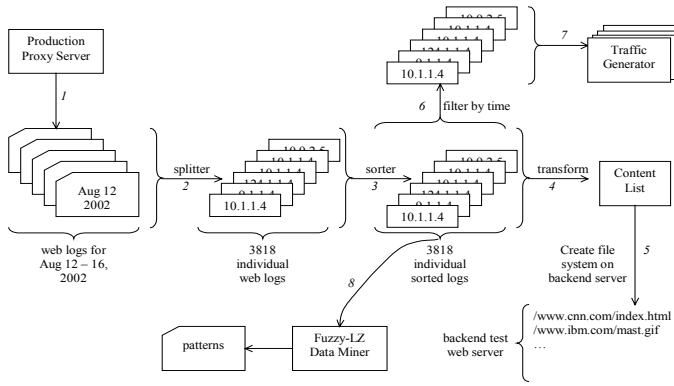

Fig. 6. Web log transformation

Step 2: Using a simple java program, we parsed and split the five log files into 3818 individual log files. Each individual log file contained all the URL requests originating from a specific client IP address[7]. As shown in Figure 6, the individual log files were named after the associated client IP address.

Step 3: Each individual log file was then sorted in ascending order by the log entry timestamp.

Step 4: By parsing all the individual log files we were able to create a file containing a list of unique file names that correspond to each unique URL. Since URL's contain characters that cannot appear in a file name or path, this step requires some transformation on the URL (ie. '%', '#', etc). We sought to develop a file system whose paths and filenames were as close as possible to the paths and filenames given in the requested URLs. In most cases, this meant we simply replaced invalid characters with a valid character. Some URLs were longer than the maximum allowed length of a path and filename on the file system; in these cases, we simply truncated the path and name.

Step 5: We replicated the file system specified in *Content List* on the testbed backend web server. Each file in the content list was reproduced in both size and file path on the web server. The content of each file was simply randomly generated bytes. By using soft links, we were able to save space by sharing files of equal size. In the end, our backend server emulated 32,169 internet web sites, contained 76,033 shared random data files, and 1,885,812 soft links representing the set of unique URL's found in the original proxy log files.

Step 6: Having built a shadow file system on the backend test web server, we ran the set of sorted individual web log files through a filter to obtain all the

[7] We identify user's with client IP addresses. We are aware of the limitations of this approach due when IP addresses are dynamically allocated using DHCP. This issue can be resolved using DHCP server logs that tracks not only the assigned IP addresses but also the the media access control address used by the network adapter hardware of the client.

requests for a specific time interval (ie. 12 noon to 1pm on Thursday, August 15th, 2002). This created a set of individual web log files that represented the behavior of each client during the specified time period.

Step 7: Finally, the traffic generator running on the client machine, would fork a process for each of the individual web log files. Each process would read its associated web log file and emulate the behavior of that client by creating the traffic pattern specified in the file. The traffic generator we used was a slightly modified version of the traffic generator described in [19], which is an improved version of the Webstone 2.5 web traffic generator [28]. In addition to the modifications described in [19], we modified the traffic generator to create traffic patterns from individual web log files.

Step 8: As described in Section 3, the Fuzzy-LZ Data Miner reads the individual sorted logs and generates access patterns.

Prefetching System Flow:
Our prefetching system is asynchronous. The prefetching occurs in two phases. The first phase does not use the prefetching module. It corresponds to the standard proxy request loop. All client HTTP requests are intercepted by the Web Proxy. In this first phase, this proxy attempts to serve these requests from its cache. If a requested document is not in the cache, the request is sent to the Web Server. Once the requested content is obtained either from the cache or from the Web Server, it is sent back to the client. The proxy also keeps track of a list of client IP addresses for which prefetching should be performed. After serving the request to the client, the proxy starts the second phase. In this phase, the proxy verifies if the IP address of this client is in the list of clients for which prefetching should be done, in which case it notifies the Prefetch Pattern Generator by sending to it the client IP address and the URL requested. At this time, the proxy ends the processing of this request. The Prefetch Pattern Generator verifies its list of access patterns for this user and sends the corresponding prefetching requests to the Proxy Server. These requests are normal HTTP requests with a special identifier placed in the "From" field of the HTTP request header. When the Web Proxy receives such requests, it identifies them as prefetching requests and passes them to a prefetching module that tries to get the requested content from the Web Server and stores it in the cache.

4.2 Measurements and Results

In this section, we describe the results of our experiments to show that our prefetching system yields an improved cache hit rate, lowers latency, and does not incur significant bandwidth overheads. These results are shown via three plots. The first plot shows the cache hit gain for 10 users spanning the entire spectrum of learnability values. The second plot shows the latency measurements for a "typical" user. A typical user is one whose learnability measure falls on the mode of the distribution, i.e. around 0.75 as seen in Figure 3. The third plot shows the prefetching overhead in terms of the number of requests per second

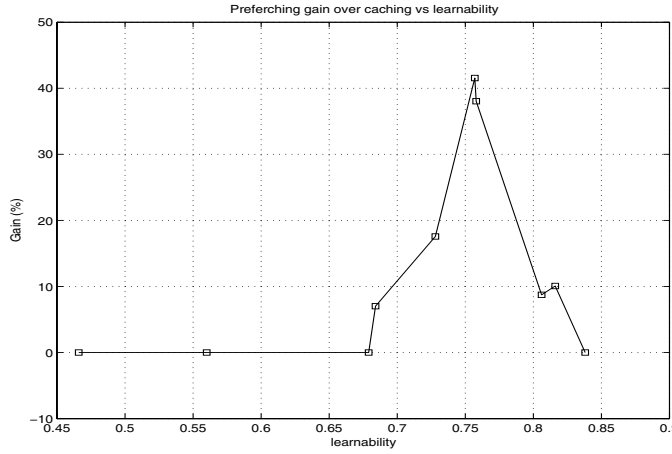

Fig. 7. Cache hit rate gain as a function of learnability

for the typical user. We define the cache hit rate gain as 1 minus the ratio of the prefetching hit rate to the pure caching hit rate. In the first plot, Figure 7, the cache hit rate gain is plotted as a function of the learnability measure. When users are very predictable, i.e., high learnability measures of ≥ 0.84, the cache hit rate gain is negligible. In other words, the cache hit rate of prefetching matches that of caching. This is expected, since predictable users tend to access the same URLs over and over and caching is designed for these scenarios. On the other hand, when users are very random, i.e., low learnability measures of ≤ 0.65, the cache hit rate gain is also negligible. In this case, the lack of identifiable patterns prevents the prefetching system from learning useful access patterns. However, in the middle, i.e., learnability measures between 0.65 and 0.85 where we have more than 80% of the users (see Figure 3), the cache hit rate gain is significant. We believe that within this interval the learnability measure is high enough that the prefetching algorithm can identify sufficient patterns such that it can achieve a non-trivial gain. At the same time, the learnability measure is not so high that the gain is offset by the ability of simple caching systems to benefit from highly predictable user access behaviors.

In the remaining two plots, we track the activity of our typical user (as described above). We performed two experiments: a baseline experiment with only pure caching, and a prefetching experiment with prefetching turned ON. As shown in Table 1, over the course of a one hour interval, prefetching increased the number of cache hits for our user from 86 hits to 122 hits. The latency is measured on the client machine.

In the second plot, Figure 8, the delay (in milliseconds) is plotted against the elapsed time. This plot shows the details of the latency for each request in this time period. The "+" markers indicate each request that resulted in a cache miss during the prefetching experiment. For these requests, the page had to be retrieved from the original server. The "×" markers indicate each request that

Table 1. Experiment summary for a typical user with prefetching ON

Learnability	0.757
Test duration	2PM to 3PM, Aug 15 2002
Total Number of requests	196
Number of cache hits (baseline)	86
Number of cache hits (prefetching)	122
Average delay of cached content (×)	32.02 ms
Average delay of prefetched content (o)	38.11 ms
Average delay of content from the Web server (+)	1747.7 ms

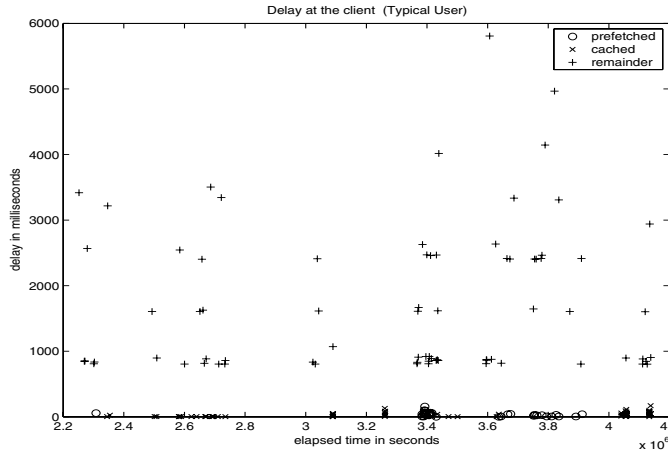

Fig. 8. Latency for each request made by the typical user

resulted in a cache hit during the baseline experiment. The "o" markers indicate all the requests that were not served from the cache in the baseline experiment but were served from the cache in the prefetching experiment. As expected, the latency for those requests served from the cache is less than the latency for those requests requiring an access to the Web Server. Note that the standard deviation for the delay of the "+" markers (server series) is 1127.8 milliseconds while the standard deviation of the delay of the "o" markers (prefetch series) is only 32.7 milliseconds. The standard deviation of the delay of the "×" markers (cached series) is 31.3 milliseconds.

In the third plot, Figure 9, the number of requests per second is plotted against the elapsed time for the prefetch, cached and server series. This plot shows the load that is placed on the proxy server. No more than 4 additional prefetching requests per second are served from the cache for our typical user. The additional load on the network is also low. We measured that 467 extra requests were made by the prefetching module in anticipation of 196 requests made by the user. In other words, for each user requests, there was in average 2.38 prefetching requests issued by the Prefetching Module.

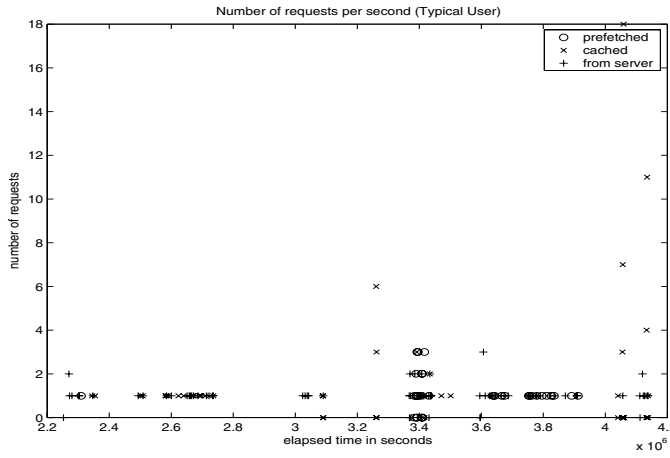

Fig. 9. Number of Requests per second for a typical user

5 Concluding Remarks

This paper has introduced Fuzzy-LZ, a new technique that learns the access patterns of individual users by mining their past access histories. We have shown that this technique predicts user access patterns in proportion to the amount of randomness within a user's access behavior. We have derived from complexity measures a metric called *learnability* that measures the amount of randomness within a user's access behavior. We have measured the learnability of a large-scale system (with 623 users) and found that most users have good learnability (at least 80% of the users have a learnability measure between 0.65 and 0.85 on a scale of 0 to 1). We have implemented and deployed the Fuzzy-LZ learning technique as the predictor within a web content prefetching system. We have seen that the learnability of a user correlates with the overall cache hit rate obtained within the system.

Our conclusion of the gain of Fuzzy-LZ based prefetching over conventional caching can be described in three cases. First, when the access behavior of users is very predictable, the cache hit rate of prefetching matches that of caching. This is not surprising, since predictable users tend to access the same content over and over and caching is designed for this case. Second, when the access behavior of users is very random, the cache hit rate of prefetching also matches that of caching. In this case, the lack of identifiable patterns prevents the prefetching system from learning useful access patterns. Third, when the access behavior of users is neither very predictable nor random, the cache hit rate of prefetching exceeds that of caching by up to 41.5 % with an overhead of only 2.4 prefetched pages per user request for a typical user. Fortunately, the large majority of users in our workload fall within this category.

There are several avenues for future work. First, we plan to perform further evaluations with additional workloads as well as various cache sizes. Second, we

believe that we can easily apply the Fuzzy-LZ technique to dynamic content. For example, we can extract general descriptions of the learned clusters and use these descriptions to pre-subscribe users to dynamic content. Finally, we plan to explore the applicability of this technique in pervasive computing environments. Within these environments, user mobility will mean that content cached in one location may not be accessed from the same location, and the increasing heterogeneity of access devices means that content formatted for one device may not be suitable for access by other devices.

References

1. A. Bestavros, Using Speculation to Reduce Server Load and Service Time on the WWW, Proceedings of the 4th ACM International Conference on Information and Knowledge Management, Baltimore, MD, 1995
2. E. Cohen,B. Krishnamurthy and J. Rexford, Efficient Algorithms for Predicting Requests to Web Servers, INFOCOM (1):284–293, 1999
3. E. Cohen and H. Kaplan, Prefetching the Means for Document Transfer: A New Approach for Reducing Web Latency, INFOCOM (2):854–863, 2000
4. T. Cover and J. Thomas, Elements of Information Theory, J. Wiley & Son, 1991
5. S. Das and D. Cook and A. Bhattacharya and E. Heierman III and T.-Y. Lin, The Role of Prediction Algorithms in the MavHome Smart Home Architecture, to appear in IEEE Personal Communications Special Issue on Smart Homes
6. B. Davison, Predicting Web Actions from HTML Content, Proceedings of the The Thirteenth ACM Conference on Hypertext and Hypermedia (HT'02) 2002
7. D. Duchamps, Prefetching Hyperlinks, Proceedings of USITS'99: The 2nd USENIX Symposium on Internet Technologies and Systems, October 1999
8. M. Feider, N. Merhav and M. Gutman, Universal Prediction of Individual Sequences, IEEE Transactions on Information Theory (38):1258-1270, 1992
9. Z. Jiang and L. Kleinrock, An adaptive network prefetch scheme, IEEE Journal on Selected Areas in Communications, 16(3):358–368, April 1998
10. B. Krishnamurthy and J. Rexford, Web Protocols and Practice, Addison Wesley, 2001
11. T. M. Kroeger, D. D. E. Long and J. C. Mogul, Exploring the Bounds of Web Latency Reduction from Caching and Prefetching, USENIX Symposium on Internet Technologies and Systems (1997),
12. A. Lempel and J. Ziv, On the Complexity of Finite Sequences, IEEE Transactions on Information Theory (22):75–81, 1976
13. M. Li and P. Vitanyi, An Introduction to Kolmogorov Complexity and its Aplications, second edition, Springer Verlag (1997)
14. Logging Control In W3C httpd, "http://www.w3.org/Daemon/User/Config/Logging.html"
15. J. Kerkhofs and K. Vanhoof and D. Pannemans, Web Usage Mining on Proxy Servers: A Case Study, Workshop on Data Mining For Marketing Applications, September 2001
16. T. Luczak and W. Szpankowski, A Suboptimal Lossy Data Compression Based on Approximate Pattern Matching, IEEE Trans. Inf. Theory, (43):1439–1451, 1997
17. T.M. Mitchell, Machine Learning, Mc Graw-Hill (1997)
18. PeakJet2000 Software, "http://www.peak.com/peakjet2long.html"

19. D. Olshefski, J. Neih and D. Agrawal, Inferring Client Response Time at the Web Server, SIGMETRICS Proceedings 160–171, Marina Del Rey, CA, June 2002
20. T. Palpanas and A. Mendelzon, Web Prefetching Using Partial Match Prediction, Proceedings of the 4th International Web Caching Workshop, 1998
21. J. Rissanen, World Scientific, Stochastic Complexity in Statistical Inquiry, 1989
22. L. Rizzo, Dummynet: a simple approach to the evaluation of network protocols, ACM SIGCOMM Computer Communication Review, 27(1):31–41, January 1997
23. J. S. Vitter and P. Krishnan, Optimal prefetching via data compression, Journal of the ACM 43(5) 771–793, 1996
24. J. Srivastava, R. Cooley, M. Deshpande and P. Tan, Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data, SIGKDD Explorations 1(2):12–23, 2000,
25. J. Van Leeuwen, (ed.): Computer Science Today. Recent Trends and Developments. Lecture Notes in Computer Science, Vol. 1000. Springer-Verlag, Berlin Heidelberg New York (1995)
26. P. Vitanyi and M. Li, Minimum Description Length Induction, Bayesianism, and Kolmogorov Complexity, IEEE Trans. on Info. Theory (46):446–464, 2000
27. The Web Collector, "http://www.inkey.com/save30/"
28. WebStone, "http://www.mindcraft.com/"
29. Q. Yang, H.H. Zhang and I.T.Y. Li, Mining web logs for prediction models in WWW caching and prefetching, in Knowledge Discovery and Data Mining, 473–478, 2001
30. O. Zaiane, M. Xin and J. Han, Discovering Web Access Patterns and Trends by Applying OLAP and Data Mining Technology on Web Logs, IEEE Advances in Digital Libraries Conference, 1998
31. J. Ziv and A. Lempel, A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, (23):337–343, 1977